Preferential theory revision

Pierangelo Dell'Acqua*† and Luís Moniz Pereira†

* Department of Science and Technology - ITN
Linköping University, 601 74 Norrköping, Sweden
pier@itn.liu.se

† Centro de Inteligência Artificial - CENTRIA
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
lmp@di.fct.unl.pt

Abstract. Employing a logic program approach, this paper focuses on applying preferential reasoning to theory revision, both by means of preferences among existing theory rules, and by means of preferences on the possible abductive extensions to the theory. And, in particular, how to prefer among plausible abductive explanations justifying observations.

1 Introduction

Logic program semantics and procedures have been used to characterize preferences among the rules of a theory [5]. Whereas the combination of such rule preferences with program updates and the updating of the preference rules themselves [4] have been tackled, a crucial ingredient has been missing, namely the consideration of abductive extensions to a theory, and the integration of revisable preferences among such extensions. The latter further issue is the main subject of this paper.

We take a theory expressed as a logic program under stable model semantics, already infused with preferences between rules, and we add a set of abducibles constituting possible extensions to the theory, governed by conditional priority rules amongst preferred extensions. Moreover, we cater for minimally revising the preferential priority theory itself, so that a strict partial order is always enforced, even as actual preferences are modified by new incoming information. This is achieved by means of a diagnosis theory on revisable preferences over abducibles, and its attending procedure.

First we supply some epistemological background to the problem at hand. Then we introduce our preferential abduction framework, and proceed to apply it to exploratory data analysis. Next we consider the diagnosis and revision of preferences, theory and method, and illustrate it on the data exploration example. Finally, we exact general epistemic remarks on the approach.

1.1 Preferences, Rationality, Theory Revision, and AI

(I) The theoretical notions of *preference* and *rationality* with which we are most familiar are those of the economists'. *Economic preference* is a comparative

choice between outcomes alternative outcomes whereby a *rational* (economic) *agent* is one whose expressed preferences over a set of outcomes exhibits the structure of a complete pre-order.

However, preferences themselves may change. Viewing this phenomena as a comparative choice, however, entails that there are meta-level preferences whose outcomes are various preference rankings of beliefs and that an agent chooses a change in preference based upon a comparative choice between the class of first-order preferences [6]. But this is an unlikely model of actual change in preference, since we often evaluate changes—including whether to abandon a change in preference–based upon items we learn *after a change in preference is made*. Hence, a realistic model of preference change will not be one that is couched exclusively in decision theoretic terms. Rather, when a conflict occurs in updating beliefs by new information, the possible items for revision should include both the set of conflicting beliefs *and* a reified preference relation underlying the belief set. The reason for adopting this strategy is that we do not know, *a priori*, what is more important—our data or our theory. Rather, as Isaac Levi has long advocated [13], rational inquiry is guided by pragmatic considerations not a priori constraints on rational belief. On Levi's view, all justification for change in belief is pragmatic in the sense that justification for belief fixation and change are rooted in strategies for promoting the goals of a given inquiry. Setting these parameters for a particular inquiry fixes the theoretical constraints for the inquiring agent. The important point to stress here is that there is no conflict between theoretical and practical reasoning on Levi's approach, since the prescriptions of Levi's theory are *not* derived from minimal principles of rational consistency or coherence [13].

(II) Suppose your scientific theory predicts an observation, o, but you in fact observe $\neg o$. The problem of carrying out a principled revision of your theory in light of the observation $\neg o$ is surprisingly difficult. One issue that must be confronted is what the principle objects of change are. If theories are simply represented as sets of sentences and prediction is represented by material implication, then we are confronted with *Duhem's Problem* [7]: If a theory entails an observation for which we have disconfirming evidence, logic alone won't tell you which among the conjunction of accepted hypotheses to change in order to restore consistency. The serious issue raised by Duhem's problem is whether disconfirming evidence targets the items of a theory in need of revision in a principled manner.

The AGM [1] conception of belief change differs to Duhem's conception of the problem in two important respects. First, whereas the item of change on Duhem's account is a set of sentences, the item of change on the AGM conception is a belief state, represented as a pair consisting of a logically closed set of sentences (a belief set) and a selection function. Theories are not represented as by replacing entailment by the AGM postulates. What remains in common is what Sven Hansson [10] has called the *input-assimilating* model of revision, whereby the object of change is a set of sentences, the input item is a particular sentence, and the output is a new set of sentences. But one insight to emerge is that the

input objects for change may not be single sentences, but a sentence-measure pair [14], where the value of the measure represents the entrenchment of the sentence and thereby encodes the ranking of this sentence in the replacement belief set [14, 18, 19]. But once we acknowledge that items of change are not *belief simpliciter* but belief and order coordinates, then there are two potential items for change: the acceptance or rejection of a belief and the change of that belief in the ordering. Hence, implicitly, the problem of preference change appears here as well.

Within the AGM model of belief change, belief states are the principal objects of change: propositional theory (belief set) changed according to what Sven Hansson [10] has called the *input-assimilating* model, whereby the object of change (a belief set) is exposed to an input (a sentence) and yields a new belief set.

(III) Computer science has adopted logic as its general foundational tool, while Artificial Intelligence *AI* has made viable the proposition of turning logic into a *bona fide* computer programming language. At the same time, *AI* has developed logic beyond the confines of monotonic cumulativity, typical of the precise, complete, endurable, condensed, and closed mathematical domains, in order to open it up to the non-monotonic real world domain of imprecise, incomplete, contradictory, arguable, revisable, distributed, and evolving knowledge. In short, *AI* has added dynamics to erstwhile statics. Indeed, classical logic has been developed to study well-defined, consistent, and unchanging mathematical objects. It thereby acquired a static character. *AI* needs to deal with knowledge in flux, and less than perfect conditions, by means of more dynamic forms of logic. Too many things can go wrong in an open non-mathematical world, some of which we don't even suspect. In the real world, any setting is too complex already for us to define exhaustively each time. We have to allow for unforeseen exceptions to occur, based on new incoming information. Thus, instead of having to make sure or prove that some condition is not present, we may assume it is not (the Closed World Assumption - CWA), on condition that we are prepared to accept subsequent information to the contrary, i.e. we may assume a more general rule than warranted, but must henceforth be prepared to deal with arising exceptions.

Much of this has been the focus of research in logic programming, a field of *AI* which uses logic directly as a programming language[1], and provides specific implementation methods and efficient working systems to do so[2]. Logic programming is moreover much used as a staple implementation vehicle for logic approaches to *AI*.

[1] Cf. [12] for the foundations and [3] for a relevant background for this paper.

[2] For a most advanced system, incorporating recent theory and implementation developments, see the XSB system at: http://xsb.sourceforge.net/.

2 Framework

2.1 Language

Let $\mathcal{L}$ be a first order language. A domain literal in $\mathcal{L}$ is a domain atom A or its default negation *not* A, the latter expressing that the atom is false by default (CWA). A domain rule in $\mathcal{L}$ is a rule of the form:

$$A \leftarrow L_1, \ldots, L_t \quad (t \geq 0)$$

where A is a domain atom and $L_1, \ldots, L_t$ are domain literals. To express preference information, $\mathcal{L}$ contains priority rules. Let $N = \{n_{r_1}, \ldots, n_{r_k}\}$ be a name set containing a unique name for every domain rule in $\mathcal{L}$. Given a domain rule r, we write n_r to indicate its name. A priority atom is an atom of the form $n_r < n_u$, where $\{n_r, n_u\} \subseteq N$.[3] $n_r < n_u$ means that rule r is preferred to rule u. We assume that names in N do not include "$<$" itself. A priority rule in $\mathcal{L}$ is a rule of the form:

$$n_r < n_u \leftarrow L_1, \ldots, L_t \quad (t \geq 0)$$

where $n_r < n_u$ is a priority atom and every L_i $(1 \leq i \leq t)$ is a domain literal or a priority literal.

We use the following convention. Given a rule r of the form $L_0 \leftarrow L_1, \ldots, L_t$, we write $H(r)$ to indicate L_0, $B(r)$ to indicate the conjunction $L_1, \ldots, L_t$. We write $B^+(r)$ to indicate the conjunction of all positive literals in $B(r)$, and $B^-(r)$ to indicate the conjunction of all negated literals in $B(r)$. When $t = 0$ we write the rule r simply as L_0.

Let $\mathcal{A} \subseteq \mathcal{L}$ be a set of domain atoms, called *abducibles*. Abducibles may be thought of as hypotheses that can be used to extend the current theory of the agent, in order to provide hypothetical solutions or possible explanations for given queries.

A (logic) program P over $\mathcal{L}$ is a finite set of domain rules and priority rules. Every program P has associated a set of abducibles $\mathcal{A}_P$, without rules in P. A 2-valued interpretation M of $\mathcal{L}$ is any set of literals from $\mathcal{L}$ that satisfies the condition that, for any atom A, precisely one of the literals A or *not* A belongs to M. We say that an interpretation M satisfies a conjunction of literals $L_1, \ldots, L_t$, and write $M \models L_1, \ldots, L_t$, if every literal L_i in the conjunction belongs to M.

2.2 Declarative Semantics

In the remaining of this section we let P be a program over $\mathcal{L}$, $\mathcal{A}_P$ the set of abducibles of P, and M an interpretation of $\mathcal{L}$. We write $least(P)$ to indicate the least model of P. We adopt the first two definitions from [9], and Definitions 4 and 5 from [4].

[3] In order to establish the preferred abductive stable models (cf. Def. 6), we require the relation induced by $<$ to be a well-founded, strict partial ordering on N.

Definition 1. *The set of default assumptions of P with respect to M is:*

$$Default(P, M) = \{not\ A : \nexists r \in P \text{ such that } H(r) = A \text{ and } M \models B(r)\}.$$

Definition 2. *M is a stable model of P iff $M = least(P \cup Default(P, M))$.*

Definition 3. *Let $\triangle \subseteq \mathcal{A}_P$. M is an abductive stable model with hypotheses $\triangle$ of P iff:*

$$M = least(P^+ \cup Default(P^+, M)),\ where\ P^+ = P \cup \triangle.$$

Note that the abducibles in $\mathcal{A}_P$ are defined false by default whenever they are not abduced. Given a program P, to compute which of its abductive stable models are preferred according to the priority relation $<$, we remove (from the program) all the unsupported rules together with the less preferred rules defeated by the head of some more preferred one, in a priority rule. Unsupported rules are those whose head is true in the model but whose body is defeated by the model, ie. some of its default negated atoms are false in it:

Definition 4. *The set of unsupported rules of P and M is:*

$$Unsup(P, M) = \{r \in P : M \models H(r), M \models B^+(r) \text{ and } M \not\models B^-(r)\}.$$

Definition 5. *$Unpref(P, M)$ is a set of unpreferred rules of P and M iff:*

$$Unpref(P, M) = least(Unsup(P, M) \cup Q),\ where$$

$$Q = \{\, r \in P :\ \exists u \in (P - Unpref(P, M))\ such\ that\ M \models n_u < n_r,\ M \models B^+(u),$$
$$and\ [\ not\ H(u) \in B^-(r)\ or\ (not\ H(r) \in B^-(u),\ M \models B(r))\ ]\ \}.$$

A rule r is unpreferred if it is unsupported or there exists a more preferred rule u (which is not itself unpreferred) such that the positive literals in $B(u)$ hold, and r is defeated by u or r attacks (i.e., attempts to defeat) u. Note that only domain rules can be unpreferred since it is required that $M \models n_u < n_r$ holds, where n_u and n_r are names of domain rules.

The following definition introduces the notion of preferred abductive stable model. Given a program P and a set $\triangle$ of hypotheses, a preferred abductive stable model with hypotheses $\triangle$ of P is a stable model of the program that contains all the hypotheses in $\triangle$, and all those rules in P that are not unpreferred.

Definition 6. *Let $\triangle \subseteq \mathcal{A}_P$ and M an abductive stable model with hypotheses $\triangle$ of P. M is a preferred abductive stable model with hypotheses $\triangle$ of P iff:*

1. *if $M \models n_{r_1} < n_{r_2}$, then $M \not\models n_{r_2} < n_{r_1}$*
2. *if $M \models n_{r_1} < n_{r_2}$ and $M \models n_{r_2} < n_{r_3}$, then $M \models n_{r_1} < n_{r_3}$*
3. *$M = least(P^+ - Unpref(P^+, M) \cup Default(P^+, M))$, with $P^+ = P \cup \triangle$.*

Conditions 1 and 2 state that the preference relation '$<$' is required to be a strict partial order. When the language contains only domain rules and priority rules (that is, there are no abducibles), the semantics reduces to the Preferential semantics of Brewka and Eiter [5]. If integrity constraints are introduced, this semantics generalizes to the Updates and Preferences semantics of Alferes and Pereira [4], which extends updatable logic programs with updatable preferences. Our semantics takes the latter, without formally addressing updating, and complements it with modifiable abducibles.

Definition 7. *An abductive explanation for a query G is any set $\triangle \subseteq \mathcal{A}_P$ of hypotheses such that there exists a preferred abductive stable model M with hypotheses $\triangle$ of P for which $M \models G$.*

A program may have several abductive explanations for a query G.

3 Preferring Abducibles

In our framework we defined the preference relation over domain rules. A possible question is: Can we also express preferences over abducibles? Being able to do so will allow us to compare the competing explanations for an observed behaviour. The evaluation of alternative explanations is one of the central problems of abduction. Indeed, an abductive problem of a reasonable size (for example in diagnosis) may have a combinatorial explosion of possible explanations to handle. Thus, it is important to generate only the explanations that are relevant for the problem at hand. Several proposals about how to evaluate competing explanations have been proposed. Some of them involve a "global" criterion against which each explanation as a whole can be evaluated. A general drawback of those approaches is that global criteria are generally domain independent and computationally expensive. An alternative to global criteria for competing alternative assumptions is to allow the theory to contain rules encoding domain specific information about the likelihood that a particular assumption be true. In our approach we can express preferences among abducibles to discard the unwanted assumptions in context. Preferences over alternative abducibles can be coded into cycles over default negation, and preferring a rule will break the cycle in favour of one abducible or another. In our framework, we employ the notion of expectation to express the preconditions for assuming an abducible a. If we have an expectation for a, and we do not have an expectation for *not* a, then a can be confirmed, and therefore a can be assumed.

To express preference criteria among abducibles, we introduce the language $\mathcal{L}^*$. A relevance atom is one of the form $a \lhd b$, where a and b are abducibles. $a \lhd b$ means that the abducible a is more relevant than the abducible b. A relevance rule is a rule of the form:

$$a \lhd b \leftarrow L_1, \ldots, L_t \quad (t \geq 0)$$

where $a \lhd b$ is a relevance atom and every L_i $(1 \leq i \leq t)$ is a domain literal or a relevance literal. Let $\mathcal{L}^*$ be a language consisting of domain rules and relevance rules.

Example 1. Consider a situation where an agent Claire drinks either tea or coffee (but not both). Suppose that Claire prefers coffee over tea when sleepy. This situation can be represented by a program Q over $\mathcal{L}^*$ with the set of abducibles $\mathcal{A}_Q = \{tea, coffee\}$:

$$Q = \left\{ \begin{array}{l} drink \leftarrow tea \\ drink \leftarrow coffee \\ expect(tea) \\ expect(coffee) \\ expect\_not(coffee) \leftarrow blood\_pressure\_high \\ coffee \lhd tea \leftarrow sleepy \end{array} \right\}$$

Notice that $\lhd$ expresses relevance among abducibles that are alternative. Therefore, Q need not contain the two rules:

$$tea \leftarrow not\ coffee \quad \text{and} \quad coffee \leftarrow not\ tea$$

expressing that the abducibles *tea* and *coffee* exclude one another. Having the notion of expectation allows one to express the preconditions for an expectation or otherwise about an assumption a, and express which possible expectations are *confirmed* (or go through) in a given situation. If the preconditions do not hold, then expectation a cannot be confirmed, and therefore a will not be assumed. For example, the rules:

$$expect(tea) \leftarrow have\_tea \quad \text{and} \quad expect(coffee) \leftarrow have\_coffee$$

express that one has an expectation for tea and coffee if he has them. By means of *expect_not* one can express situations where one does not expect something. The rule

$$expect\_not(coffee) \leftarrow blood\_pressure\_high$$

states not to expect coffee if one has high blood pressure. In this case, coffee will not be confirmed or go through because of the contrary expectation arising as well (and therefore tea will be assumed). ▯

The following definition exploits the relevancy relation $\lhd$ of a program Q to distinguish which of its abductive stable models are relevant.

Definition 8. *Let Q be a program over $\mathcal{L}^*$ with set of abducibles $\mathcal{A}_Q$ and M an interpretation of $\mathcal{L}^*$. Let $a \in \mathcal{A}_Q$ be an abducible. M is a relevant abductive stable model of Q with hypothesis $\triangle = \{a\}$ iff:*

1. *for every $x, y \in \mathcal{A}_Q$, if $M \models x \lhd y$ then $M \not\models y \lhd x$*
2. *for every $x, y, z \in \mathcal{A}_Q$, if $M \models x \lhd y$ and $M \models y \lhd z$, then $M \models x \lhd z$*
3. *there exists no relevance rule r in Q such that $M \models B(r)$, $H(r) = x \lhd a$, $M \models expect(x)$, and $M \not\models expect\_not(x)$*
4. *$M \models expect(a)$ and $M \not\models expect\_not(a)$*

5. $M = least(Q^+ \cup Default(Q^+, M))$, *with* $Q^+ = Q \cup \triangle$.

Letting $\triangle$ be a singleton guarantees that the abducibles in $\mathcal{A}_Q$ are alternative in the sense that only one can be assumed. Note that for simplicity of exposition, we consider $\triangle$ to be a singleton. This can be generalized to a set of abducibles, and the preference order can be adapted to one among sets. As required by a preference relation, it is natural to demand that the relevancy relation be a strict partial order (conditions 1 and 2 above). Condition 3 guarantees that there exists no abducible x (which can be confirmed) more relevant than a. The notion of expectation is incorporated directly into the definition of relevant abductive stable model by condition 4. Finally, condition 5 requires interpretation M to be an abductive stable model with hypotheses $\triangle$.

Example 2. Let Q be the program of Example 1. Q has two alternatives explanations $\triangle_1 = \{coffee\}$ and $\triangle_2 = \{tea\}$ for the query *drink*. In fact, Q has two relevant abductive stable models:

$$M_1 = \{expect(tea), expect(coffee), coffee, drink\} \text{ with hypotheses } \triangle_1$$

$$M_2 = \{expect(tea), expect(coffee), tea, drink\} \text{ with hypotheses } \triangle_2$$

for which $M_1 \models drink$ and $M_2 \models drink$. The number of models reduces to one if we add *sleepy* to Q. In this case, *coffee* being an abducible more relevant than *tea* and consequently the only relevant model of Q is $M_1 \cup \{sleepy\}$. ▯

The following syntactical transformation maps programs over $\mathcal{L}^*$ into programs over $\mathcal{L}$, and thereby gives us a proof procedure for the language $\mathcal{L}^*$.

Definition 9. *Let Q be a program over $\mathcal{L}^*$ with set of abducibles $\mathcal{A}_Q = \{a_1, \ldots, a_m\}$. The program $P = \Sigma(Q)$ with abducibles $\mathcal{A}_P = \{abduce\}$ is obtained as follows:*

1. *P contains all the domain rules in Q;*
2. *for every $a_i \in \mathcal{A}_Q$, P contains the domain rule:*
   $confirm(a_i) \leftarrow expect(a_i), not\ expect\_not(a_i)$
3. *for every $a_i \in \mathcal{A}_Q$, P contains the domain rule:*
   $a_i \leftarrow abduce, not\ a_1, \ldots, not\ a_{i-1}, not\ a_{i+1}, \ldots, not\ a_m, confirm(a_i) \quad (r_i)$
4. *for every relevance rule $a_i \lhd a_j \leftarrow L_1, \ldots, L_t$ in Q, P contains the priority rule:*
   $r_i < r_j \leftarrow L_1, \ldots, L_t$

To take into consideration expectations, the transformation Σ adds (step 2) a rule defining the notion of confirmation for every abducible a_i in $\mathcal{A}_Q$. Then, Σ codes the alternative abducibles of $\mathcal{A}_Q$ into mutually defeating cycles over default negation (step 3), and preferring a rule (step 4) will break the cycle in favour of one abducible or another. Note that every rule added at step 3 contains in its body the abducible *abduce* and *confirm*(a_i). The role of *abduce* is to enact the assumption of one of the alternative assumptions needed to prove the query[4], while the role of *confirm*(a_i) is to require that the expectations for a_i are satisfied. It is easy to see that $\Sigma(Q)$ is a program over the language $\mathcal{L}$.

[4] If the query holds without making assumptions, then *abduce* is not abduced and none of the alternative assumptions in $\mathcal{A}_Q$ can be assumed.

Example 3. Let Q be the program of Example 1. The transformation Σ maps Q into the program P with abducibles $\mathcal{A}_P = \{abduce\}$:

$$P = \left\{ \begin{array}{ll} drink \leftarrow tea & \\ drink \leftarrow coffee & \\ & \\ expect(tea) & \\ expect(coffee) & \\ & \\ expect\_not(coffee) \leftarrow blood\_pressure\_high & \\ & \\ coffee \leftarrow abduce, not\ tea, confirm(coffee) & (1) \\ tea \leftarrow abduce, not\ coffee, confirm(tea) & (2) \\ & \\ confirm(tea) \leftarrow expect(tea), not\ expect\_not(tea) & \\ confirm(coffee) \leftarrow expect(coffee), not\ expect\_not(coffee) & \\ & \\ 1 < 2 \leftarrow sleepy & \end{array} \right\}.$$

The role of the abducible *abduce* is to enact the assumption of one of the alternative assumptions *tea* or *coffee* needed to prove *drink*. The rules (1) and (2) code the alternative assumptions *tea* and *coffee* into cycles over negation. Rule (1) says that *coffee* can be assumed if *abduce* has been abduced, *tea* is not assumed, and coffee is confirmed. The last rule in P is a priority rule stating that rule (1) is preferable to rule (2) if *sleepy* holds. P has two preferred abductive stable models with hypotheses $\triangle = \{abduce\}$:

$$M_1 = \{\ abduce, confirm(tea), confirm(coffee), expect(tea), expect(coffee), coffee, drink\ \}$$

$$M_2 = \{\ abduce, confirm(tea), confirm(coffee), expect(tea), expect(coffee), tea, drink\ \}$$

The number of preferred abductive stable models reduces to one if *sleepy* holds. In that case, the unique preferred abductive stable model would be:

$$M_3 = \{\ abduce, confirm(tea), confirm(coffee), expect(tea), expect(coffee), coffee, drink, sleepy, 1{<}2\ \}.$$ □

The following result states the correctness of the transformation Σ. Given an interpretation M, we write $\widehat{M}$ to indicate the interpretation obtained from M by removing the abducible *abduce*, the priority atoms, and all the domain atoms of the form *confirm*(.).

Proposition 1. *Let Q be a program over $\mathcal{L}^*$ with abducibles $\mathcal{A}_Q$ and $P = \Sigma(Q)$. Then, M is a preferred abductive stable model with hypotheses $\triangle = \{abduce\}$ of P iff $\widehat{M}$ is a relevant abductive stable model of Q.*

4 Exploratory Data Analysis

Another application of expressing preferences over abducibles is that of *exploratory data analysis*, which aims at suggesting a pattern for further inquiry,

and contributes to the conceptual and qualitative understanding of a phenomenon. Assume that a unexpected phenomenon, x, is observed by an agent Bob, and that Bob has three possible hypotheses (abducibles) a, b, c, capable of explaining it. In exploratory data analysis, after observing some new facts, we abduce explanations and explore them to check predicted values against observations. Though there may be more than one convincing explanation, we abduce only the more plausible. The next example illustrates explanatory data analysis.

Example 4. Let the program Q over $\mathcal{L}^*$, with abducibles $\mathcal{A}_Q = \{a, b, c\}$, be the theory of agent Bob:

$$Q = \left\{ \begin{array}{l} x \leftarrow a \\ x \leftarrow b \\ x \leftarrow c \\ \mathit{expect}(a) \\ \mathit{expect}(b) \\ \mathit{expect}(c) \\ a \lhd c \leftarrow \mathit{not}\ e \\ b \lhd c \leftarrow \mathit{not}\ e \\ b \lhd a \leftarrow d \end{array} \right\}$$

where the meaning is as follows:
x - the car does not start,
a - the battery has problems,
b - the ignition is damaged,
c - there is no gasoline in the car,
d - the car's radio works,
e - the wife has used the car, and
exp - test if the car's radio works.

Q has two relevant abductive stable models capable of explaining observation x:

$M_1 = \{\mathit{expect}(a), \mathit{expect}(b), \mathit{expect}(c), a \lhd c, b \lhd c, a, x\}$ with hypothesis $\triangle_1 = \{a\}$

$M_2 = \{\mathit{expect}(a), \mathit{expect}(b), \mathit{expect}(c), a \lhd c, b \lhd c, b, x\}$ with hypothesis $\triangle_2 = \{b\}$

In this example, we have only a partial relevancy theory over abducibles. Thus, we cannot select exactly one abducible (i.e., one model), as it were the case had we a complete relevancy relation over all abducibles in $\mathcal{A}_Q$. To prefer between a and b, one can perform some experiment exp to obtain confirmation (by observing the environment) about the most plausible hypothesis. To do so, we can employ active rules that are rules of the form:

$$L_1, \ldots, L_t \Rightarrow \alpha : A$$

where $L_1, \ldots, L_t$ are domain literals, and $\alpha : A$ is an action literal. This rule states to update the theory of an agent α with A if its body $L_1, \ldots, L_t$ is satisfied in all relevant abductive stable models. For example, we can add the following rules (where env plays the role of the environment) to the theory Q of Bob:

$$\left\{ \begin{array}{l} \mathit{choose} \leftarrow a \\ \mathit{choose} \leftarrow b \end{array} \right\} \text{ together with } \left\{ \begin{array}{l} a \Rightarrow \mathit{Bob} : \mathit{chosen} \\ b \Rightarrow \mathit{Bob} : \mathit{chosen} \\ \mathit{choose} \Rightarrow \mathit{Bob} : (\mathit{not}\ \mathit{chosen} \Rightarrow \mathit{env} : \mathit{exp}) \end{array} \right\}.$$

Initially Bob has two hypotheses, a and b, that are capable of explaining the observed phenomena x. Hence, Bob must discover the correct one. Bob chooses some hypothesis if a or b hold:

$$\begin{array}{l} \mathit{choose} \leftarrow a \\ \mathit{choose} \leftarrow b. \end{array}$$

With this knowledge, *Bob* still has two relevant abductive stable models: $M_3 = M_1 \cup \{choose\}$ and $M_4 = M_2 \cup \{choose\}$. As *choose* holds in both models, the last active rule is triggerable. When triggered, it will add (at the next state) the active rule $not\ chosen \Rightarrow env : exp$ to the theory of *Bob*, and, if *not chosen* holds, *Bob* will perform the experiment *exp*. The first two active rules are needed to prevent *Bob* from performing *exp* when *Bob* has chosen one of the abducibles. ▯

5 Revising Relevancy Relations

The relevancy relation $\lhd$ is required by Definition 8 to be a strict partial order. Relevancy relations are subject to be modified when new information is brought to the knowledge of an individual, or aggregated when we need to represent and reason about the simultaneous relevancy relations of several individuals. The resulting relevancy relation may not be a strict partial order and must therefore be revised. This section investigates the problem of revising relevancy relations by means of declarative debugging. Mark that, more generally, the conditions on the preference order need not be those of a strict partial order, but may be any other desirable conditions. In any case, the resulting possible revisions provide as many alternative coherent choices on the preferences actually adopted as a result of any single revision.

Example 5. Let $\lhd_1$ and $\lhd_2$ be two relevancy relations. Suppose that $\lhd_1$ and $\lhd_2$ are combined by boolean composition, that is, $\lhd = \lhd_1 \cup \lhd_2$. Clearly, $\lhd$ is not a strict partial order being antisymmetric, and transitivity not being preserved. Consider the following program Q over $\mathcal{L}^*$ with abducibles $\mathcal{A}_Q = \{a, b, c\}$:

$$Q = \left\{ \begin{array}{l} x \leftarrow a \\ x \leftarrow b \\ x \leftarrow c \\ expect(a) \\ expect(b) \\ expect(c) \end{array} \right\} \cup \left\{ \begin{array}{l} \bar{u} \lhd \bar{v} \leftarrow \bar{u} \lhd_1 \bar{v} \\ \bar{u} \lhd \bar{v} \leftarrow \bar{u} \lhd_2 \bar{v} \\ a \lhd_1 b \\ b \lhd_1 c \\ b \lhd_2 a \end{array} \right\}$$

where $\bar{u}$ and $\bar{v}$ are variables ranging over the abducibles in $\mathcal{A}_Q$. The program Q does not have any relevant abductive stable model since $\lhd$ is not a strict partial order and therefore conditions 1 and 2 of Definition 8 are not met. ▯

With the aim of revising relevancy relations, we introduce the language $\mathcal{L}^+$ extending $\mathcal{L}^*$ to contain integrity constraints. The latter are rules of the form:

$$\bot \leftarrow L_1, \ldots, L_t \quad (t \geq 0)$$

where $\bot$ is a domain atom denoting contradiction, and $L_1, \ldots, L_t$ are domain or relevance literals. Integrity constraints are rules that enforce some condition, and in this case they take the form of denials. Domain rules are distinct from integrity constraints and must not be expressed as denials. In domain rules, it is of crucial importance which atom occurs in their head. The language $\mathcal{L}^+$ consists

of domain rules, relevance rules, and integrity constraints. In $\mathcal{L}^+$ there are no abducibles, and therefore its meaning is characterized in terms of stable models. Given a program T over $\mathcal{L}^+$ and a literal L, we write $T \models L$ if L is true in every stable model of T. The program T is contradictory if $T \models \bot$. Clearly, programs over $\mathcal{L}^+$ are liable to be contradictory because of the integrity constraints.

We introduce now the notion of diagnosis, adapted from [16], to handle relevancy relations. Given a contradictory program T, to revise its contradiction ($\bot$) we have to modify T by adding and removing rules. In this framework, the diagnostic process reduces to finding such rules. To specify which rules in T may be added or removed, we assume given a set C of predicate symbols of $\mathcal{L}^+$. C induces a partition of T into two disjoint parts: a changeable one T_c and a stable one T_s. The part T_c contains the rules in T defining atoms in C, while T_s contains the rules in T defining atoms not belonging to C. The part T_c is the one subject to the diagnosis process.

Definition 10. *Let T be a program and C a set of predicate symbols in $\mathcal{L}^+$. Let D be a pair $\langle U, I\rangle$ where $U \subseteq C$ and $I \subseteq T_c$. Then D is a diagnosis for T iff $(T - I) \cup U \not\models \bot$. The pair $\langle \{\}, \{\}\rangle$ is called the empty diagnosis.*

Intuitively, a diagnosis specifies the rules to be added and removed from the changeable part of T to revise its contradiction $\bot$. In order to minimize the number of changes we consider minimal diagnoses.

Definition 11. *Let T be a program and $D = \langle U, I\rangle$ a diagnosis for T. Then, D is a minimal diagnosis for T iff there exists no diagnosis $D_2 = \langle U_2, I_2\rangle$ for T such that $(U_2 \cup I_2) \subseteq (U \cup I)$.*

The following example illustrates the notion of minimal diagnosis. To check whether or not the relevancy relation $\lhd$ of a program is a strict partial order, we need to express (within the program itself) the properties required for $\lhd$.

Example 6. Consider the program Q of Example 5. To express that the relevancy relation of Q is a strict partial order, we add to Q the rules:

$$T = Q \cup \left\{ \begin{array}{l} \bot \leftarrow \bar{u} \lhd \bar{u} \\ \bot \leftarrow \bar{u} \lhd \bar{v}, \bar{v} \lhd \bar{u} \\ \bot \leftarrow \bar{u} \lhd \bar{v}, \bar{v} \lhd \bar{z}, \mathit{not}\ \bar{u} \lhd \bar{z} \end{array} \right\}$$

where $\bar{u}, \bar{v}$, and $\bar{z}$ are variables ranging over the abducibles in $\mathcal{A}_Q$. Since $a \lhd b$ and $b \lhd a$ belong to every stable model of T, we conclude $\bot$ and thus engender a contradiction. To revise T we need to identify its stable and changeable part. Let $C = \{\lhd_1, \lhd_2\}$. This means that only the relevancy relations $\lhd_1$ and $\lhd_2$ are subject to revision:

$$T_c = \left\{ \begin{array}{l} a \lhd_1 b \\ b \lhd_1 c \\ b \lhd_2 a \end{array} \right\} \quad \text{and} \quad T_s = T - T_c. \quad T \text{ admits three minimal diagnoses:}$$

$D_1 = \langle \{\}, \{a \lhd_1 b\}\rangle$, $D_2 = \langle \{\}, \{b \lhd_1 c, b \lhd_2 a\}\rangle$ and $D_3 = \langle \{a \lhd_1 c\}, \{b \lhd_2 a\}\rangle$. ▯

To compute the minimal diagnoses of a contradictory program T, we employ the contradiction removal method presented in [16], adapted here to handle relevancy relations. The contradiction removal method is based on the idea of revising (to false) some of the default atoms *not* A. A default atom *not* A can be revised to false by simply adding A to T. According to [16] the default literals *not* A that are allowed to change their truth value are those for which there exists no rule in T defining A. Such literals are called revisable.

Definition 12. *The revisables of a program T over $\mathcal{L}^+$ is a subset of the set of atoms A (with $A \neq \bot$) for which there are no rules defining A in T.*

Definition 13. *Let T be a program over $\mathcal{L}^+$ and V a set of revisables of T. A set $Z \subseteq V$ is a revision of T iff $T \cup Z \not\models \bot$.*

Example 7. Consider the contradictory program $T = T_s \cup T_c$:

$$T_s = \left\{ \begin{array}{l} \bot \leftarrow a, a' \\ \bot \leftarrow b \\ \bot \leftarrow d, not\ f \end{array} \right\} \quad \text{and} \quad T_c = \left\{ \begin{array}{l} a \leftarrow not\ b, not\ c \\ a' \leftarrow not\ d \\ c \leftarrow e \end{array} \right\}$$

with revisables $V = \{b, d, e, f\}$. Intuitively the literals *not* b, *not* d and *not* e are true by CWA, entailing a and a', and hence $\bot$ via the first integrity constraint. The revisions of T are $\{e\}$, $\{d, f\}$, $\{e, f\}$ and $\{d, e, f\}$, where the first two are minimal. ∎

The following transformation maps programs over $\mathcal{L}^+$ into equivalent programs that are suitable for contradiction removal.

Definition 14. *Let T be a program over $\mathcal{L}^+$ and C a set of predicate symbols in $\mathcal{L}^+$. The transformation Γ that maps T into a program T' is obtained by applying to T the following two operations:*

- *Add not incorrect$(A \leftarrow Body)$ to the body of each rule $A \leftarrow Body$ in T_c.*
- *Add the rule $p(\bar{x}_1, \ldots, \bar{x}_n) \leftarrow uncovered(p(\bar{x}_1, \ldots, \bar{x}_n))$ for each predicate p with arity n in C, where $\bar{x}_1, \ldots, \bar{x}_n$ are variables.*

We assume the predicate symbols *incorrect* and *uncovered* do not occur in T. The following result states the correctness of Γ.

Theorem 1. Let T be a program over $\mathcal{L}^+$ and L a literal. Then $T \models L$ iff $\Gamma(T) \models L$.

Example 8. Let T be the program of Example 6. Then, the program $\Gamma(T)$ is:

$$\Gamma(T) = \left\{ \begin{array}{l} x \leftarrow a \\ x \leftarrow b \\ x \leftarrow c \\ expect(a) \\ expect(b) \\ expect(c) \\ \\ \bar{u} \lhd \bar{v} \leftarrow \bar{u} \lhd_1 \bar{v} \\ \bar{u} \lhd \bar{v} \leftarrow \bar{u} \lhd_2 \bar{v} \end{array} \right\} \cup \left\{ \begin{array}{l} \bot \leftarrow \bar{u} \lhd \bar{u} \\ \bot \leftarrow \bar{u} \lhd \bar{v}, \bar{v} \lhd \bar{u} \\ \bot \leftarrow \bar{u} \lhd \bar{v}, \bar{v} \lhd \bar{z}, not\ \bar{u} \lhd \bar{z} \\ \\ a \lhd_1 b \leftarrow not\ incorrect(a \lhd_1 b) \\ b \lhd_1 c \leftarrow not\ incorrect(b \lhd_1 c) \\ b \lhd_2 a \leftarrow not\ incorrect(b \lhd_2 a) \\ \\ \bar{u} \lhd_1 \bar{v} \leftarrow uncovered(\bar{u} \lhd_1 \bar{v}) \\ \bar{u} \lhd_2 \bar{v} \leftarrow uncovered(\bar{u} \lhd_2 \bar{v}) \end{array} \right\}$$

The minimal revisions of $\Gamma(T)$ with respect to the revisables of the form incorrect(.) and uncovered(.) are:

$$Z_1 = \{incorrect(a \lhd_1 b)\}$$
$$Z_2 = \{incorrect(b \lhd_1 c), incorrect(b \lhd_2 a)\}$$
$$Z_3 = \{uncovered(a \lhd_1 c), incorrect(b \lhd_2 a)\}$$

It is easy to see that Z_3, for instance, is a revision of $\Gamma(T)$ since the unique stable model M of $\Gamma(T) \cup Z_3$ is:

$$M = \{\, a \lhd c, a \lhd b, b \lhd c, a \lhd_1 c, a \lhd_1 b, b \lhd_1 c, expect(a), expect(b), expect(c),$$
$$uncovered(a \lhd_1 c), incorrect(b \lhd_2 a)\}$$

and $M \not\models \bot$. ∎

The following result relates the minimal diagnoses of a program T with the minimal revisions of $\Gamma(T)$.

Theorem 2. Let T be a program over $\mathcal{L}^+$. The pair $D = \langle U, I \rangle$ is a diagnosis for T iff

$$Z = \{uncovered(A) : A \in U\} \cup \{incorrect(A \leftarrow Body) : A \leftarrow Body \in I\}$$

is a revision of $\Gamma(T)$, where the revisables are all the literals of the form *incorrect*(.) and *uncovered*(.). Furthermore, D is a minimal diagnosis iff Z is a minimal revision.

To compute the minimal diagnosis of a program T we consider the transformed program $\Gamma(T)$ and compute its minimal revisions. An algorithm for computing minimal revisions in such logic programs is given in [16].

6 Concluding Remarks

We have shown that preferences and priorities (they too a form of preferential expressiveness) can enact choices amongst rules and amongst abducibles, which are dependant on the specifics of situations, all in the context of theories and theory extensions expressible as logic programs. As a result, using available transformations provided here and elsewhere [2], these programs are executable by means of publicly available state-of-the-art systems [11]. In [2], we furthermore have shown how preferences can be integrated with knowledge updates, and how they fall too under the purview of updating, again in the context of logic programs. Preferences about preferences are also adumbrated therein.

We have employed the two-valued Stable Models semantics to provide meaning to our logic programs, but we could just as well have employed the three-valued Well-Founded Semantics [8] for a more skeptical preferential reasoning.

Also, we need not necessarily insist on a strict partial order for preferences, but have indicated that different conditions can be provided. The possible alternative revisions, required to satisfy the conditions, impart a non-monotonic or

defeasible reading of the preferences given initially. Such a generalization permits us to go beyond just a foundational view of preferences, and allows us to admit a coherent view as well, inasmuch several alternative consistent stable models may obtain for our preferences, as a result of each revision.

Other logic program semantics are available too, such as the Revised Stable Model semantics, a two-valued semantics which resolves odd loops over default negation, arising from the unconstrained expression of preferences, by means of *reductio ad absurdum* [17]. Indeed, when there are odd loops over default negation in a program, Stable Model semantics does not afford the program with a semantics.

In [18], arguments are given as to how epistemic entrenchment can be explicitly expressed as preferential reasoning. And, moreover, how preferences can be employed to determine believe revisions, or, conversely, how belief contractions can lead to the explicit expression of preferences.

[6] provides a stimulating survey of opportunities and problems in the use of preferences, reliant on AI techniques.

We advocate that the logic programming paradigm (LP) provides a well-defined, general, integrative, encompassing, and rigorous framework for systematically studying computation, be it syntax, semantics, procedures, or attending implementations, environments, tools, and standards. *LP* approaches problems, and provides solutions, at a sufficient level of abstraction so that they generalize from problem domain to problem domain. This is afforded by the nature of its very foundation in logic, both in substance and method, and constitutes one of its major assets.

Indeed, computational reasoning abilities such as assuming by default, abducing, revising beliefs, removing contradictions, preferring, updating, belief revision, learning, constraint handling, etc., by dint of their generality and abstract characterization, once developed can readily be adopted by, and integrated into, distinct topical application areas.

No other computational paradigm affords us with the wherewithal for their coherent conceptual integration. And, all the while, the very vehicle that enables testing its specification, when not outright its very implementation [15]. Consequently, it merits sustained attention from the community of researchers addressing the issues we have considered and have outlined.

Acknowledgments

We thank Gregory Wheeler for a discussion and comment on a previous draft of the paper.

References

1. Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *J. Symbolic Logic*, 50(2):510–530, 1985.

2. J. J. Alferes, P. Dell'Acqua, and L. M. Pereira. A compilation of updates plus preferences. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Logics in Artificial Intelligence*, LNAI 2424, pages 62–74, Berlin, 2002. Springer-Verlag.
3. J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming.* LNAI 1111. Springer-Verlag, 1996.
4. J. J. Alferes and L. M. Pereira. Updates plus preferences. In M. O. Aciego, I. P. de Guzmn, G. Brewka, and L. M. Pereira, editors, *Logics in AI, Procs. JELIA'00*, LNAI 1919, pages 345–360, Berlin, 2000. Springer.
5. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109:297–356, 1999.
6. Jon Doyle. Prospects for preferences. *Computational Intelligence*, 20(2):111–136, 2004.
7. Pierre Duhem. *The Aim and Structure of Physical Theory.* Princeton University Press, 2nd edition, 1954.
8. A. V. Gelder, K. A. Ross, and J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. *J. ACM*, 38(3):620–650, 1991.
9. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *ICLP'88*, pages 1070–1080. MIT Press, 1988.
10. Sven Hansson. Ten philosophical problems in belief revision. *Journal of Logic and Computation*, 13:37–49, 2003.
11. XSB-Prolog. XSB is available at xsb.sourceforge.net.
12. R. A. Kowalski. *Logic for Problem Solving.* North Holland, New York, 1979.
13. Isaac Levi. *Mild Contraction.* Clarendon Press, Oxford, 2004.
14. Nayak. Iterated belief change based on epistemic entrenchment. *Erkenntnis*, 41:353–390, 1994.
15. L. M. Pereira. Philosophical Incidence of Logic Programming. In Handbook of the Logic of Argument and Inference, D. Gabbay et al. (eds.), pp. 425-448, Studies in Logic and Practical Reasoning series, Vol. 1, Elsevier Science, 2002.
16. L. M. Pereira, C. Damásio, and J. J. Alferes. Debugging by Diagnosing Assumptions. In P. Fritzson, editor, *1st Int. Ws. on Automatic Algorithmic Debugging, AADEBUG'93*, LNCS 749, pages 58–74. Springer, 1993. Preproceedings by Linköping Univ.
17. L. M. Pereira and A. M. Pinto. Revised stable models – a semantics for logic programs. To appear in Progress in Artificial Intelligence, A. Cardoso, C. Bento, G. Dias (eds.), Procs. 12th Portuguese Intl. Conf. on Artificial Intelligence (EPIA'05), Springer, LNAI, Covilhã, Portugal, December 2005.
18. Hans Rott. *Change, Choice and Inference.* Oxford University Press, Oxford, 2001.
19. Wolfgang Spohn. Ordinal conditional functions: A dynamic theory of epistemic states. In William L. Harper and Brian Skyrms, editors, *Causation in Decision, Belief Change and Statistics*, volume 2, pages 105–134. Reidel, 1987.